COMSC-210 Lecture Topic 5
Big Data Applications

Reference
wikipedia.org

Big Data
"data sets so large or complex that traditional...
...data processing applications are inadequate"

Lab 5 Data Set: All DVC Course Offerings Since 2000
Click HERE to download input file for this program.
"flat file" format (vs SQL, JSON, AJAX)
69,201 records with tab-separated fields:
- semester (like Fall 2015)
- section# (like 8375)
- instructor (like Burns)
- date/time (like MW 7:00-9:50pm)
- room# (like L-142/149)

possible uses:
1. how many MATH courses offered in specified time?
2. who's taught COMSC-210 in the last 5 years?
3. when was ART-107 last taught?
4. how many times has Prof. Burns taught COMSC-210?
5. what's the room schedule for ATC-115 this semester?

etc.

Logistical Problems With Big Data
memory: storing records in a program
- DVC data file: 4MB
- C++ EXE stack space: 1MB

time: sorting records
- sorting 70K records = 2G compares!

Programming Issues With Data Records
`struct/class` design per record
data structure container for records
converting data source into records
- flat file parsing
- SQL database connection
- TCP/IP request/response

validation:
- how do we know results are correct?

data integrity:
- how do we know *data* is valid?

Parsing Flat Files In C++
read whole line into a `char` buffer
use C string tokenizer functions to separate fields

How To "Append" An Object To A DynamicArray

```
DynamicArray<Road> roads; // DynamicArray of ROAD objects
...
Road road; // create new ROAD object
...
roads[roads.size()] = road; // append to DynamicArray
...
for (int i = 0; i < roads.size(); i++)
  ...roads[i]...
```

How To Sort A DynamicArray

```
#include <algorithm>
...
using namespace std;
...
for (int i = 0; i < roads.size(); i++)
```

Lab 5a: Count Subject Code Offerings
for ALL records in the whole database:

```
ADJUS, 557 sections
ADS, 206 sections
AET, 62 sections
```

algorithm:
create `struct` to store SUBJECT objects with
  subject code name & section count
  and less-than operator
create DynamicArray of SUBJECT objects (initially empty)
for each parsed record:
  see if subject code matches an already-stored SUBJECT object
  if so, add one to its course count
  otherwise, create new SUBJECT object,
    set object's subject code
    set object's section count to 1
    append object to DynamicArray
sort SUBJECT objects in DynamicArray (alphabetical)
output SUBJECT objects in DynamicArray

Lab 5b: Count Course Offerings By Subject Code
for ALL records in the whole database:

```
ADJUS, 16 course(s)
  ADJUS-120, 191 section(s)
  ADJUS-121, 57 section(s)
  ADJUS-122, 40 section(s)
  ADJUS-130, 24 section(s)
  ADJUS-203, 20 section(s)
...
```

algorithm:
create `struct` to store COURSE objects with course
  name & section count
create `struct` to store SUBJECT objects with subject
  code name & DynamicArray of COURSE objects
  and less-than operator
create DynamicArray of SUBJECT objects (initially empty)
for each parsed record:
  see if subject code matches an already-stored SUBJECT object
  if so, see if its DynamicArray has a match for course
    if so, add one to its course count
    otherwise, create new COURSE object,
      set object's course name
      set object's section count to 1
      append object to DynamicArray
  otherwise, create new SUBJECT object,
    set object's subject code
    create new COURSE object,
      set object's course name
      set object's section count to 1
      append object to DynamicArray
    append SUBJECT object to DynamicArray
sort SUBJECT objects in DynamicArray (alphabetical)
output SUBJECT objects in DynamicArray
  output subject code name
  output object's Dynamic array of COURSE objects

Data Integrity
the DVC database has some *duplicate entries*
should be unique: term+section
need to track already-seen term+section combos
  skip records with already-seen term+section

```
for (int j = i + 1; j < roads.size(); j++)
  if (roads[j] < roads[i])
     swap(roads[j], roads[i]);
```

apply duplicate checking to lab 5

Road struct/class must have an operator less-than

A program that reads and parses dvc-schedule.txt

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;

#include <cstring>

int main()
{
  // for parsing the inputfile
  char* token;
  char buf[1000];
  const char* const tab = "\t";

  // open the input file
  ifstream fin;
  fin.open("dvc-schedule.txt");
  if (!fin.good()) throw "I/O error";

  // read the input file
  while (fin.good())
  {
    // read the line
    string line;
    getline(fin, line);
    strcpy(buf, line.c_str());
    if (buf[0] == 0) continue;

    // parse the line
    const string term(token = strtok(buf, tab));
    const string section(token = strtok(0, tab));
    const string course((token = strtok(0, tab)) ? token : "");
    const string instructor((token = strtok(0, tab)) ? token : "");
    const string whenWhere((token = strtok(0, tab)) ? token : "");
    if (course.find('-') == string::npos) continue; // invalid line
    const string subjectCode(course.begin(), course.begin() + course.find('-'));

    // if I get this far, then it's a valid record
    cout << term << '|' << section << '|' << course << '|' << instructor << '|' << whenWhere << '|' << subjectCode << endl;
  }
  fin.close();
}
```

Click HERE to download input file for this program.